SQL Problems - 1 for Data Analysis

1. Identifying Customers Without Orders

```sql
SELECT
    *
FROM customers
LEFT JOIN orders
ON customers.customer_id = orders.customer_id
WHERE order_id IS NULL;
```

2. Identifying Orders Without Payments

```sql
SELECT
    *
FROM orders
LEFT JOIN payments
ON orders.order_id = payments.order_id
WHERE payment_id IS NULL;
```

3. Customer Order and Payment Details

```sql
SELECT
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date,
    COALESCE(p.payment_method, 'No Payment') AS payment_method
FROM
    customers c
LEFT JOIN
    orders o
    ON c.customer_id = o.customer_id
LEFT JOIN
    payments p
    ON o.order_id = p.order_id
ORDER BY
    c.customer_id, o.order_date;
```

4. Total Sales by Customer

```sql
SELECT
    c.customer_id,
    c.customer_name,
    SUM(o.order_amount) total_sales_by_customer
FROM orders o
INNER JOIN
    customers c
    ON o.customer_id = c.customer_id
GROUP BY c.customer_id, c.customer_name
ORDER BY total_sales_by_customer DESC;
```

5. Total Payments by Customer

```sql
SELECT
    c.customer_id,
    c.customer_name,
    SUM(p.payment_amount) total_payment
FROM payments p
LEFT JOIN orders o
    ON p.order_id = o.order_id
LEFT JOIN customers c
    ON c.customer_id = o.customer_id
GROUP BY c.customer_id, c.customer_name
ORDER BY total_payment DESC;
```

6. Total Payments (INNER JOIN)

```sql
SELECT
    c.customer_id,
    c.customer_name,
    SUM(p.payment_amount) AS total_payment
FROM
    customers c
INNER JOIN
    orders o
    ON c.customer_id = o.customer_id
INNER JOIN
    payments p
    ON o.order_id = p.order_id
GROUP BY
    c.customer_id, c.customer_name
ORDER BY
    total_payment DESC;
```

7. Orders Without Payments

```sql
SELECT
    o.order_id,
    COALESCE(p.payment_amount, 0) amount
FROM orders o
LEFT JOIN payments p
    ON o.order_id = p.order_id
WHERE COALESCE(p.payment_amount, 0) = 0;
```

8. Order Payments with Pending Status

```sql
SELECT
    o.order_id,
    o.customer_id,
    p.payment_amount,
    COALESCE(p.payment_method, 'Pending') payment_method
FROM orders o
LEFT JOIN
    payments p
    ON o.order_id = p.payment_id
ORDER BY p.payment_amount DESC;
```

9. Categorizing Orders Based on Amount

```sql
SELECT
    order_id,
    order_amount,
    CASE
        WHEN order_amount > 300 THEN 'Large'
        WHEN order_amount < 300 AND order_amount > 100 THEN 'Medium'
        WHEN order_amount < 100 THEN 'Small'
    END Categorizing
FROM
    orders;
```

10. High-Value vs Regular Customers

```sql
SELECT
    c.customer_id,
    c.customer_name,
    SUM(o.order_amount) total_orders,
    CASE
        WHEN SUM(o.order_amount) > 1800 THEN 'High-Value'
        ELSE 'Regular'
    END cCat
FROM
    customers c
LEFT JOIN
    orders o
    ON c.customer_id = o.customer_id
GROUP BY
    c.customer_id, c.customer_name
HAVING SUM(o.order_amount) > 1800;
```

11. Unpaid Orders

```sql
SELECT
    o.order_id,
    o.customer_id,
    o.order_amount,
    CASE
        WHEN payment_amount IS NULL THEN 'Unpaid'
        WHEN payment_amount IS NOT NULL THEN 'Paid'
```

```
    END
FROM
    orders o
LEFT JOIN
    payments p
    ON o.order_id = p.order_id
WHERE payment_amount IS NULL;
```

12. Revenue Breakdown by State

```
SELECT
    c.state,
    SUM(o.order_amount) total_revenue,
    CASE
        WHEN SUM(o.order_amount) > 60000 THEN 'High-Revenue'
        WHEN SUM(o.order_amount) < 60000 AND SUM(o.order_amount) > 55000 THEN 'Moderate
Revenue'
        WHEN SUM(o.order_amount) < 55000 THEN 'Low Revenue'
    END 'Revenue Breakdown by State'
FROM customers c
LEFT JOIN
    orders o
    ON c.customer_id = o.customer_id
GROUP BY c.state;
```

13. Customer Lifetime Value

```
SELECT
    c.customer_id,
    COALESCE(SUM(o.order_amount), 0) 'Total Order Value',
    CASE
        WHEN SUM(o.order_amount) > 1500 THEN 'Gold'
        WHEN SUM(o.order_amount) < 1500 AND SUM(o.order_amount) > 1000 THEN 'Silver'
        WHEN SUM(o.order_amount) < 1000 THEN 'Bronze'
        ELSE 'No-medals'
    END 'Customer Lifetime Value'
FROM customers c
LEFT JOIN orders o
    ON c.customer_id = o.customer_id
GROUP BY c.customer_id
ORDER BY 'Total Order Value' DESC;
```

14. Payment Status Report

```
SELECT
    o.order_id,
    o.customer_id,
    c.customer_name,
    COALESCE(p.payment_amount, 0) payment_amount,
    CASE
        WHEN p.payment_amount IS NULL THEN 'Unpaid'
        ELSE 'Paid'
    END AS payment_status
FROM orders o
LEFT JOIN payments p
```

```
    ON o.order_id = p.order_id
LEFT JOIN customers c
    ON c.customer_id = o.customer_id
WHERE p.payment_amount IS NULL
ORDER BY
    c.customer_id, o.order_date;
```

15. Customer Status

```
SELECT
    c.customer_id,
    COALESCE(SUM(o.order_amount), 0) 'Total Revenue',
    CASE
        WHEN COALESCE(SUM(o.order_amount), 0) > 0 THEN 'Active'
        ELSE 'Inactive'
    END 'Customer Status'
FROM customers c
LEFT JOIN orders o
    ON c.customer_id = o.customer_id
GROUP BY c.customer_id
ORDER BY 'Total Revenue' DESC;
```

16. Payment Recovery Report

```
SELECT
    o.order_id,
    c.customer_id,
    c.customer_name,
    o.order_date,
    o.order_amount,
    COALESCE(SUM(p.payment_amount), 0) AS total_paid,
    (o.order_amount - COALESCE(SUM(p.payment_amount), 0)) AS amount_due,
    CASE
        WHEN SUM(p.payment_amount) IS NULL OR SUM(p.payment_amount) = 0 THEN 'Unpaid'
        WHEN SUM(p.payment_amount) < o.order_amount THEN 'Partial'
        ELSE 'Fully Paid'
    END AS payment_status
FROM
    orders o
LEFT JOIN
    customers c
    ON o.customer_id = c.customer_id
LEFT JOIN
    payments p
    ON o.order_id = p.order_id
GROUP BY
    o.order_id, c.customer_id, c.customer_name, o.order_date, o.order_amount
ORDER BY
    o.order_date;
```

17. Monthly Sales Trends

```
SELECT
    FORMAT(o.order_date, 'yyyy-MM') AS month, -- Extract year and month
    SUM(o.order_amount) AS total_sales,
```

```
    CASE
        WHEN SUM(o.order_amount) < 50000 THEN 'Low Sales'
        ELSE 'High Sales'
    END AS sales_category
FROM
    orders o
GROUP BY
    FORMAT(o.order_date, 'yyyy-MM')
ORDER BY
    month;
```